大家好，我们今天来学一下虚幻引擎的网络框架。虚幻引擎最引人注目的特点之一是其内置的网络支持。它为世界上最流行的一些在线多人游戏作支持，它可以帮助你简化多人游戏构建的过程。

## **简介**

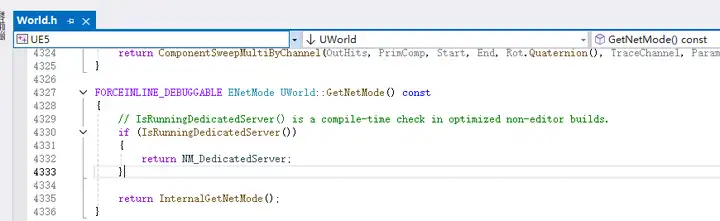
在虚幻的网络模型中，玩家连接到服务器，服务器保持游戏世界的权威状态。当服务器上发生更改时，这些更改将根据需要被“复制”到客户端。



今天我们来了解下虚幻引擎的复制系统，了解它是如何工作的，以及如何使用它。

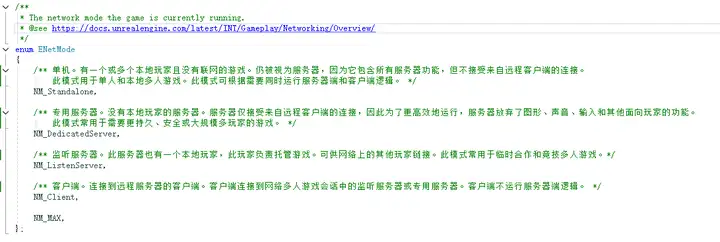
## **网络模式**

首先，让我们谈谈网络模式。网络模式是虚幻世界(UWorld)中的一个属性。



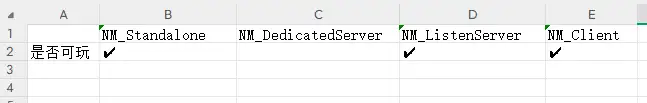
网络模式决定了游戏是在单机、客户端还是服务器环境下运行。这对于处理网络游戏中的逻辑（如同步、复制、RPC 等）非常关键。

它可以是这四个值中的任何一个。



我们可以根据三个简单的问题来区分这些网络模式。

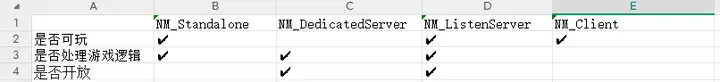
首先，我们是否有本地玩家，我们是否需要将游戏世界渲染出来？



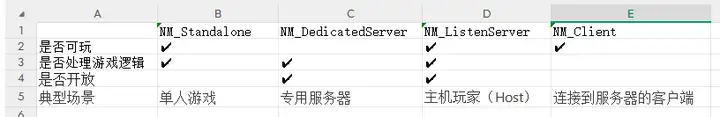
接下来，我们是否需要处理游戏逻辑？

IMG_260

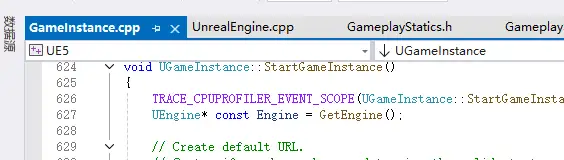
最后，我们是否对外开放，允许其他玩家进来玩？

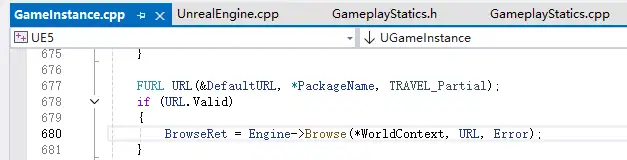


四种模式对应着这些典型的场景。



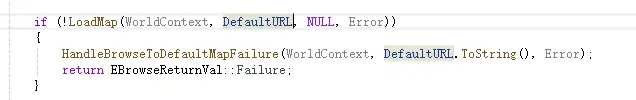
当我们启动游戏时，我们会得到一个与进程的生命周期相关的游戏实例对象。然后游戏会浏览到一个URL，这个URL可能是服务器地址或者是地图名称，游戏会用这个URL加载地图。



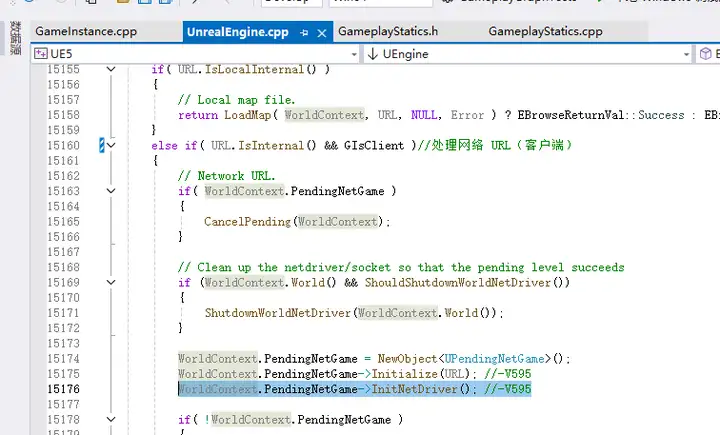


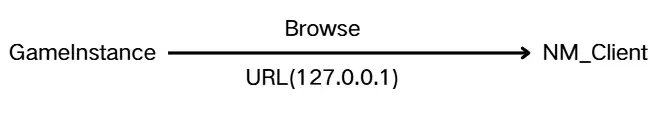
我们用不同的启动方式启动游戏，游戏的网络模式也会有所不同。Browse通过解析传入的URL来决定网络模式。



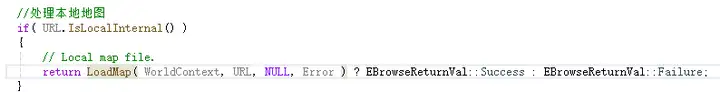


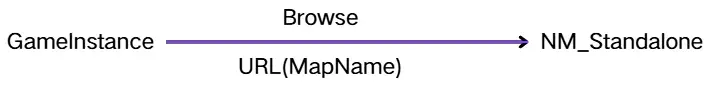
如果你的游戏实例已连接到远程服务器，那么它的网络模式会被设置为客户端模式。





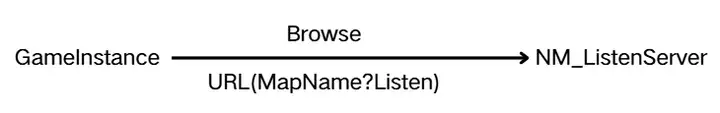
如果你的游戏实例已在本地加载了地图，则它的网络模式将是独立的。实际上，你的单个游戏实例既是服务器又是客户端，但由于它是在单个玩家配置中运行的，因此它不开放供客户端连接。





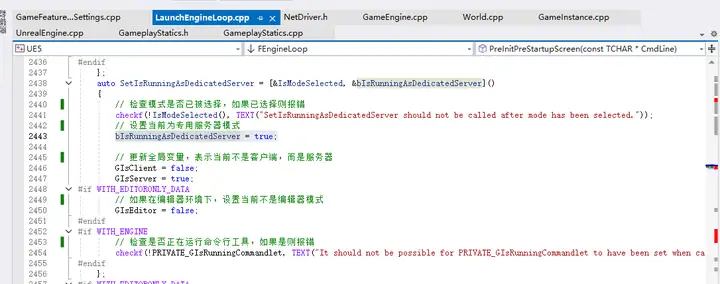
如果你在本地加载地图，但又添加了Listen选项，则你游戏实例将作为Listen服务器运行。这基本上与standalone相同，但游戏的其他实例可以作为客户端连接。

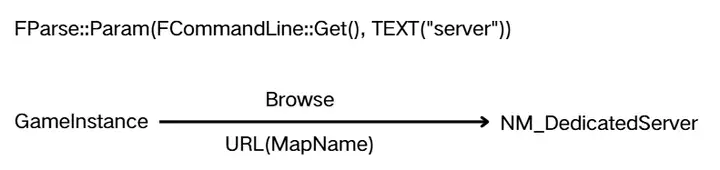




最后，你可以运行一个专用服务器。这是一个没有本地玩家和视口的游戏实例，它只是一个仅限服务器的控制台应用程序，玩家可以作为客户端连接。

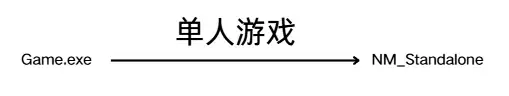
专用服务器模式通常通过命令行参数启动，如果命令行包含server参数，则进入专用服务器模式。



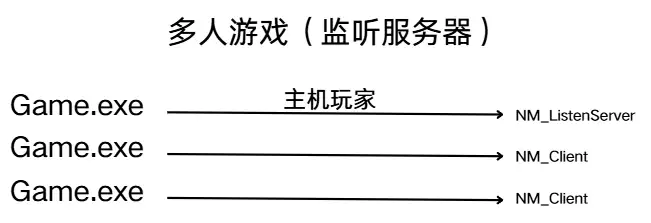


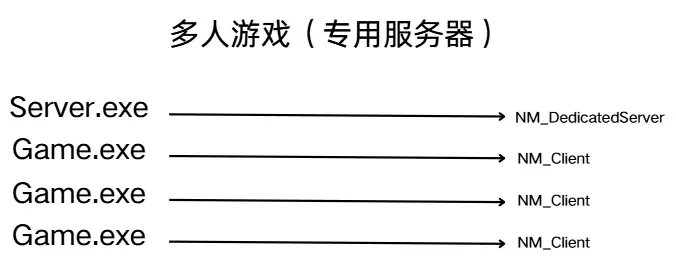
因此，虚幻游戏可以运行三种不同的网络场景。

对于单人游戏，以独立模式运行。



对于多人游戏，你有多个进程，每个进程都有自己的游戏实例。其中一个进程是监听服务器或专用服务器，其余的为客户端。





## **复制系统基础**

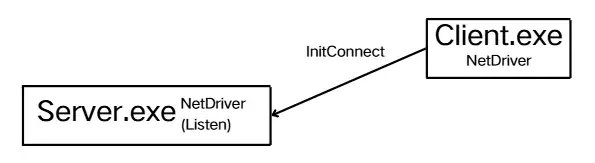
一旦我们有一个多人游戏在运行，虚幻的复制系统就会在后台工作，以确保游戏的所有这些不同的游戏实例是同步的。

为了实现这一点，复制系统依赖于三个重要的类，UNetDriver、UNetConnection和UChannel。

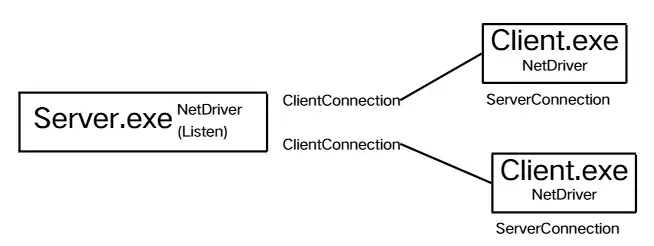
当我们启动服务器时，它会创建一个NetDriver并开始监听客户端连接。



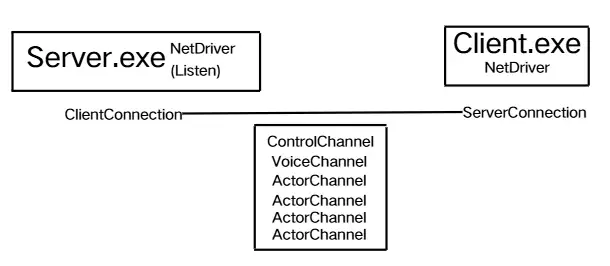
当我们启动客户端时，它会创建自己的NetDriver，该NetDriver会向服务器发送连接请求。



一旦服务器和客户端NetDriver建立联系，这将在每个NetDriver中建立网络连接，服务器将为每个客户端提供一个网络连接(NetConnection)。

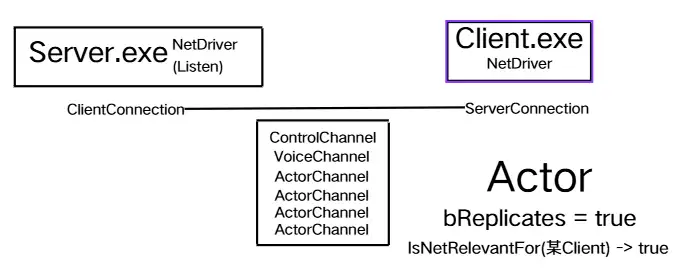


每个客户端都有一个网络连接(NetConnection)，代表它与服务器的连接。每个网络连接都有许多不同的频道(Channel)。通常，—个连接将有一个控制通道和一个语音通道，然后每个连接将有一系列Actor通道，每个Actor当前正在通过该连接复制。



这说明了复制系统的一个关键因素，即复制发生在Actor的层面上。

如果你想某Actor在网络上保持同步，你需要将该Actor设置成可复制，然后服务器将会在网络连接中打开Actor通道，服务器和客户端将使用该通道交换有关该Actor的信息。



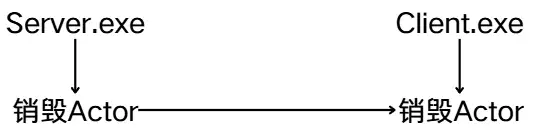
## **Actor复制**

如果Actor被复制到客户端，可能会发生三件重要的事情。

首先，Actor的生命周期会在服务器和客户端之间保持同步。如果服务器生成一个可复制的Actor，那么客户端也会收到通知，生成这个Actor的副本。



如果Actor在服务器上被销毁，它也会在客户端上被自动销毁。

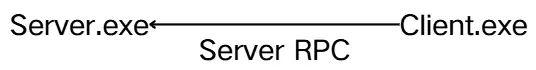


然后就是复制属性。如果Actor有一个被标记为复制的属性，那么当该属性在服务器上更改时，新值将传播到客户端。

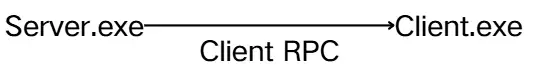


最后是RPC远程过程调用。在虚幻引擎中，RPC分为三种类型。

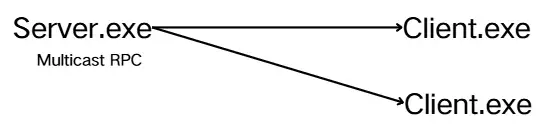
第一种是Server RPC。客户端调用，但实际执行在服务器上。用于客户端向服务器发送请求或通知。如玩家输入、技能释放、交互请求等。



第二种是Client RPC。服务器调用，但实际执行在客户端上。用于服务器向特定客户端发送通知或更新。如更新 UI、播放音效、触发特效等。

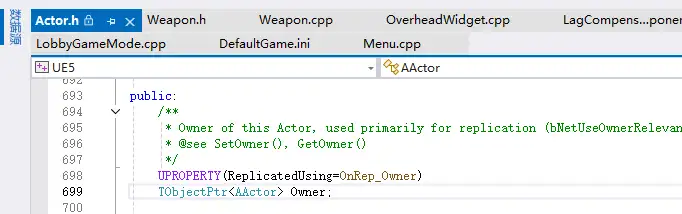


第三种是Multicast RPC。当你在服务器上调用一个Multicast RPC函数时，服务器会向所有客户端发送一条消息，指示它们在自己的Actor副本上调用该函数。如爆炸效果、全局事件通知、同步动画等。



## **所有权**

所有权是Actor复制的另一个重要概念。每个Actor都可以有一个所有者，通常是一个PlayerController或另一个 Actor。所有者决定了Actor的网络复制范围和RPC调用权限。

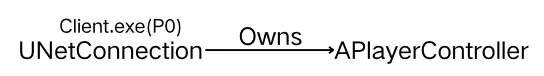


通常，你可以在Actor生成时设置所有者，但你也可以在运行时调用SetOwner设置所有者。

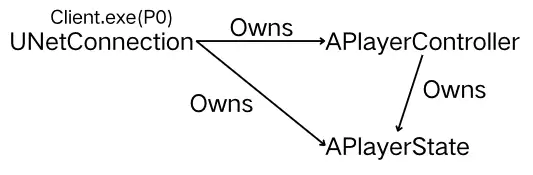


IMG_291

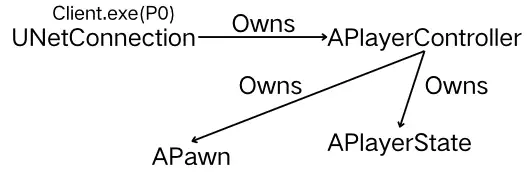
玩家控制器类（APlayerController）在所有权方面具有特殊重要性。基本上，我们的每个网络连接代表一个玩家，一旦玩家完全登录到游戏中，它就有一个与之相关的玩家控制器Actor。



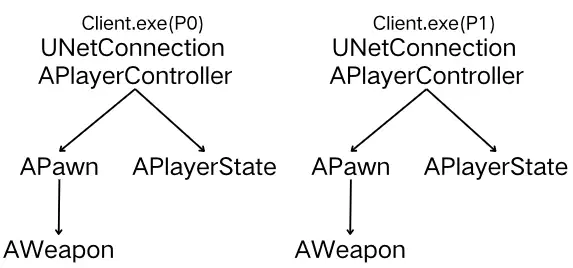
从服务器的角度来看，连接拥有该玩家控制器，进而连接拥有任何可以追溯其所有权到玩家控制器的Actor。



玩家控制器自动拥有他们所拥有的Pawn。



因此，假设我们的每个玩家控制器都有一个Pawn，每个Pawn都衍生出一个它拥有的武器actor。

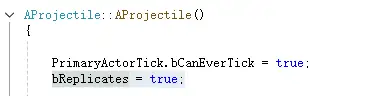


服务器可以查看这些武器actor之一，按照其所有者引用链返回玩家控制器，并确定武器actor属于特定的客户端连接。

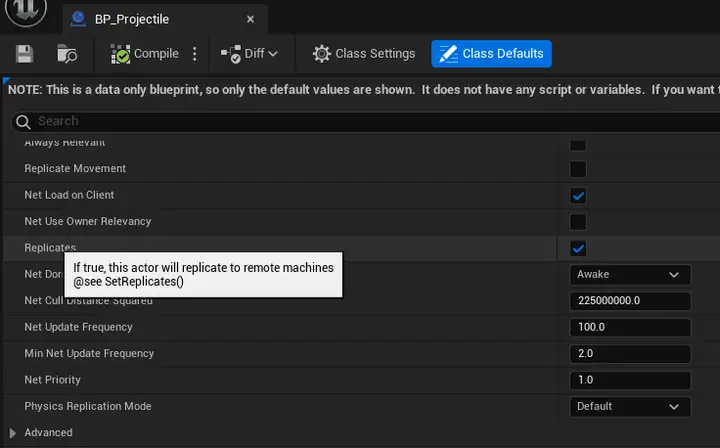
IMG_296

## **启用 Actor 复制**

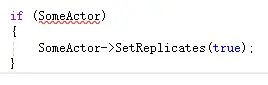
因此，让我们更仔细地看看如何配置要复制的Actor。为了让一个actor被考虑进行复制，它需要将其replicates标志设置为true。



通常，您只需在构造函数中设置此值，或者对于蓝图，您需要在类默认值中选中 “复制” 框。

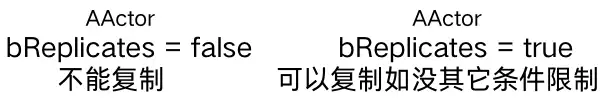


但你也可以在运行时打开或关闭复制。

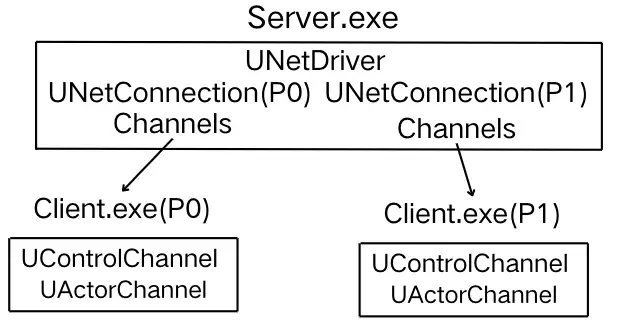




长话短说，如果一个actor启用了它的复制标志，那么它就有资格进行复制。



服务器可以在任何给定的网络连接内打开参与者通道，以便将该参与者复制到相应的客户端。

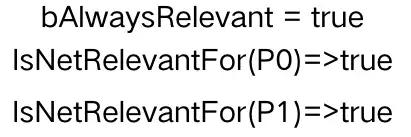


## **相关性**

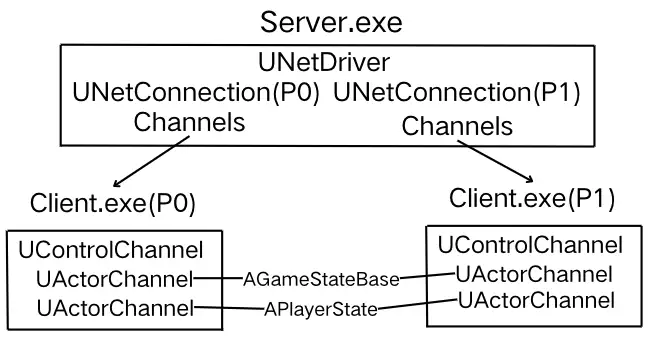
Actor的相关性决定了哪些联系会在什么时候发生。当一个Actor有资格进行复制时。服务器的网络驱动程序经常会针对每个客户端连接检查该Actor，以确定它是否与该客户端相关。

IMG_303

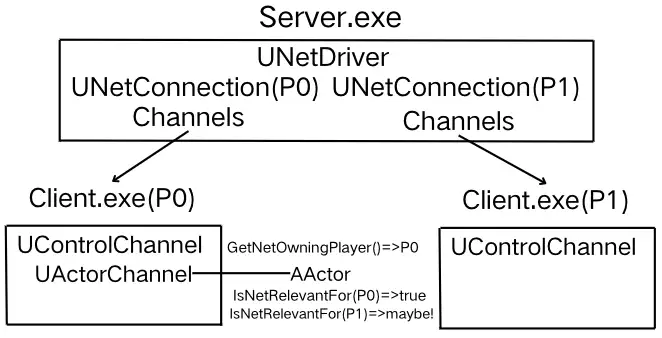
有些Actor总是相关的。这意味着只要它们符合复制条件，服务器将始终将它们复制到所有客户端。



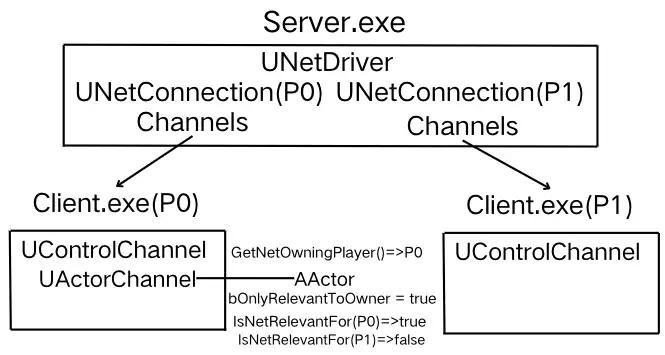
例如，游戏状态和玩家状态Actor始终是相关的。



所有权在相关性中起着重要作用。由特定玩家拥有或煽动的Actor将始终被认为与相应的客户相关。



一些actor，如玩家控制器，被配置为仅与其所有者相关，因此它们永远不会复制到非所有者客户端。



您还可以配置一个Actor，使其从所有者那里继承相关性。

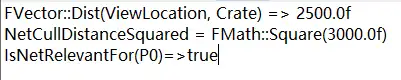
如果这些特殊标志都没有设置，并且有问题的客户端不拥有该actor，则默认行为会像这样工作。

IMG_308

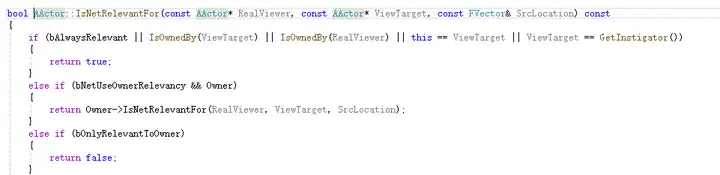
如果actor被隐藏并且其根组件已禁用碰撞，则它将不被视为相关的。

IMG_309

否则的话。相关性基于与客户端连接对应的玩家的距离，如果Actor到玩家的距离的平方小于近邻距离的平方，则Actor与该玩家相关。

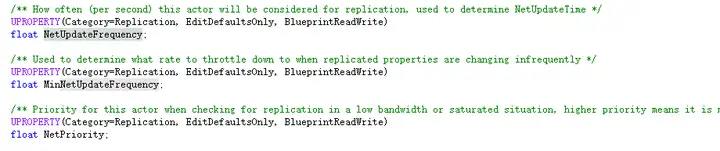


这些规则并不是硬编码的。您可以覆盖任何Actor类的函数相关的i-net，以提供您自己的自定义规则。

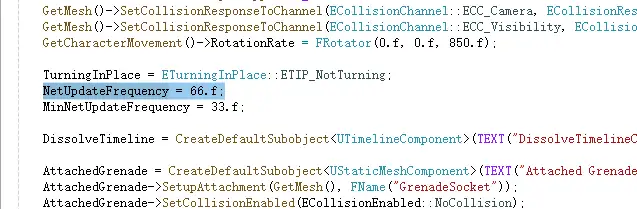


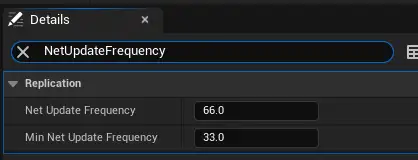
## **更新频率和优先级**

一旦复制actor，其更新频率和优先级将决定服务器向与该Actor相关的客户端发送最新进展的频率。



设置网络更新频率将决定服务器每秒检查参与者的次数，如果有任何更改，可能会向客户端发送新数据。

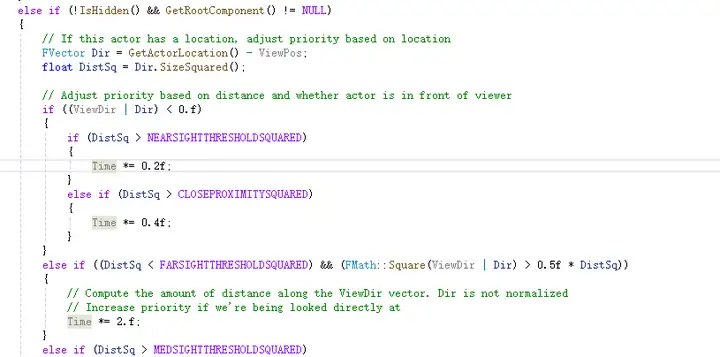




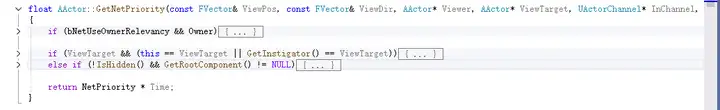
请记住，现实世界的网络延迟变化极大，带宽很快就会成为一个限制因素。因此，即使你的Actor设置为每秒更新60次，您也不会看到完美的平滑结果。

另一方面，服务器的网络驱动程序采用一些简单的负载平衡来缓解带宽饱和。在任何时间点，网络驱动程序都有有限的带宽可供使用，因此它会根据优先级对相关参与者进行排序，然后运行网络最新进展，直到用尽其可用带宽。

更接近玩家的Actor将以更高的优先级等待，而一段时间没有更新的Actor也将具有更高的优先级，因此每个Actor最终都会到达优先级列表的前面。



设置参与者的净优先级属性将会对该权重应用额外的比例。因此，例如，您可以规定一个重要Actor的更新频率应该比其他情况下多五倍。

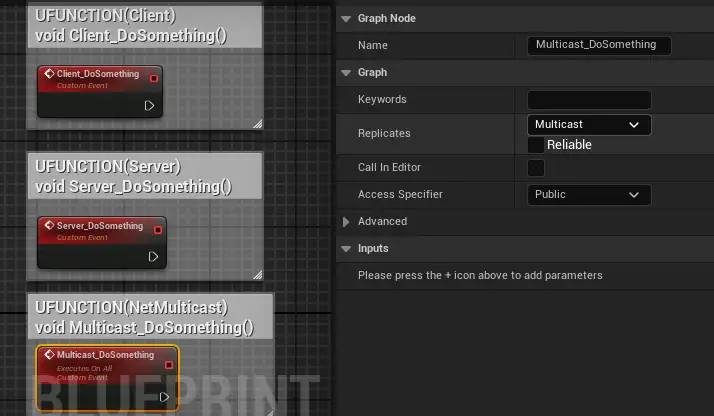


IMG_317

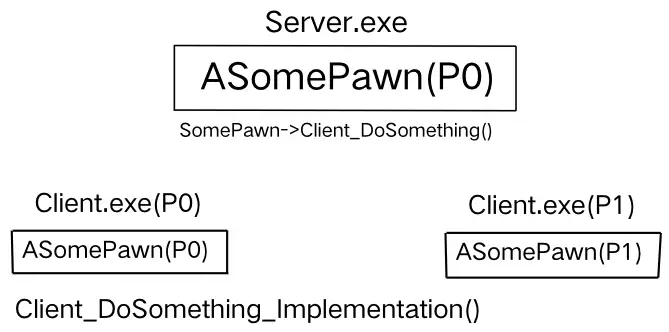
## **RPC详细信息**

定期带宽有限的网络最新进展过程主要适用于复制属性。如果您有要立即通过网络发送的高优先级消息，这就是RPCs的用途。

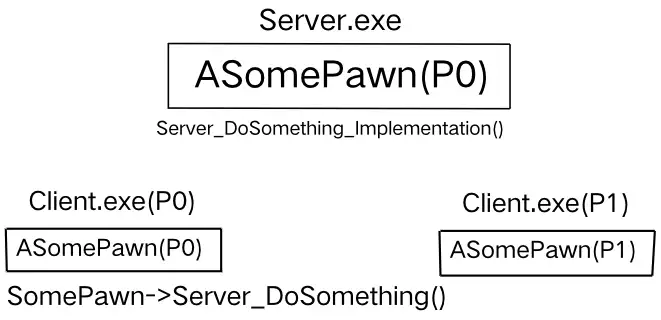
任何Ufunction都可以被指定为客户端服务器或网络多播，使其成为RPC。



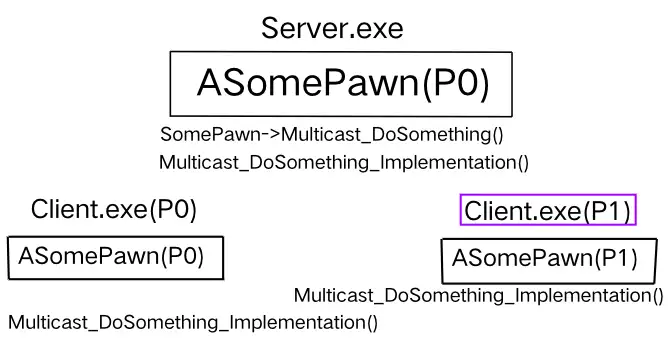
如果您在服务器上调用客户端RPC，则函数实现将在拥有的客户端上运行。



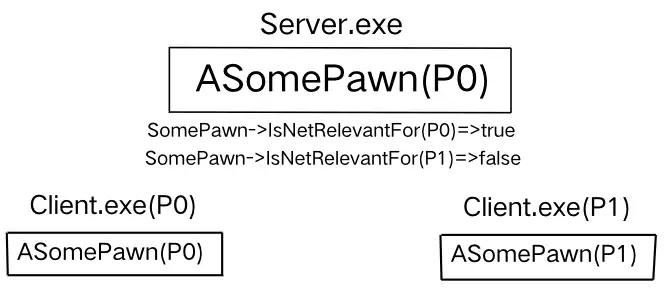
如果您从自己的和客户端调用服务器RPC，则函数实现将在服务器上运行。



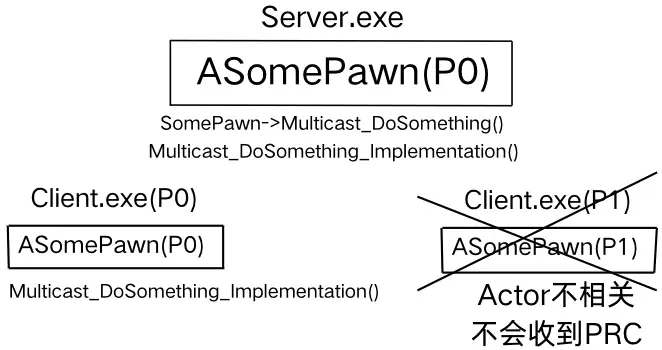
如果您从服务器调用多播RPC，则函数实现将在服务器上的每个位置以及所有客户端上运行。



与服务器和客户端RPCs不同，相关性是多播RPCs的一个因素，因为非拥有客户端可能没有对参与者开放的通道。

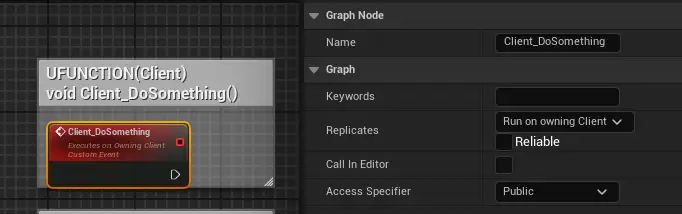


在这种情况下，客户端根本不会接收RPC。



这意味着您不应该依赖多播RPCs将持久性状态更改复制到客户端。

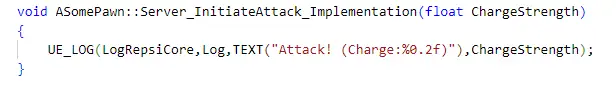
RPCs可以被宣布为可靠或不可靠的，如果带宽饱和并且它们不能保证到达，则不可靠的Rpcs可能会被丢弃，因此也不能保证它们以任何顺序到达。



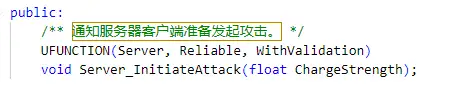
可靠的rpc保证到达，并且在单个参与者中，可靠的rpc保证按照调用的顺序到达。如果您的函数调用对游戏玩法至关重要，则这种可靠性是必要的，但如果过度使用，它可能会加剧带宽饱和，并在数据包丢失时导致瓶颈。

IMG_325

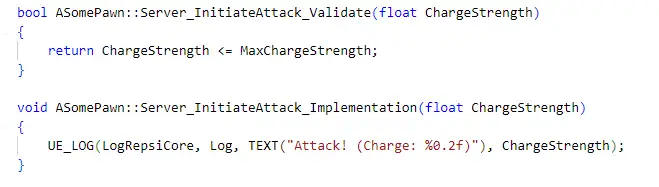
在C++中，你的函数的实际体需要用实现后缀定义。这是实际在远程进程上运行的函数，而您在本地调用的函数是一个自动生成的thunk，它通过网络服务器发送适当的消息。



Rpcs也可以使用验证进行声明，在这种情况下，您需要实现相应的验证函数，接受所有相同的参数，并返回一个布尔值，指示这些值是否可信。



这是一种作弊检测方法，适用于服务器使用从客户端发送的数据影响游戏玩法的情况。



请记住，如果服务器RPC未能通过验证，发送该RPC的客户端将被踢出游戏。

因此Rpcs会立即发送，并且它们可以是可靠的。这使得它们在某些复制属性相对有限的情况下很有用。

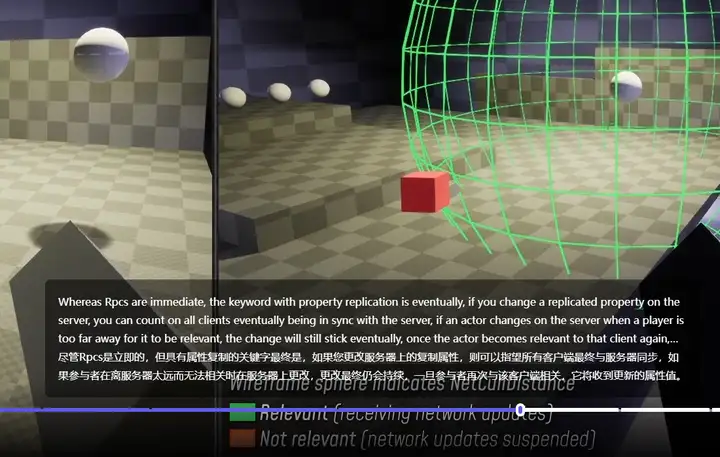
服务器Rpcs是通过拥有连接从客户端获取数据到服务器的唯一方法，因此它们有自己的位置，但总的来说，rpc被保留用于高优先级和关键时间的网络代码。例如，引擎的角色移动系统可以自由地来回使用Rpcs位置最新进展，因为移动预测和纠正需要尽可能短的延迟更新信息。

对于其他一切，你应该依靠复制的属性，无论你在哪里可以逃脱它。

## **属性复制详细信息**

属性复制是虚幻复制系统的主力，负载平衡和优先级特性使其更具可扩展性。尽管RPCs是立即的，但具有属性复制的关键字最终是。

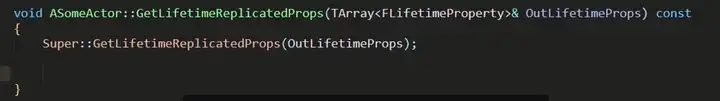
如果您更改服务器上的复制属性，则可以指望所有客户端最终与服务器同步，如果Actor在离服务器太远而无法相关时在服务器上更改，更改最终仍会持续，一旦参与者再次与该客户端相关，它将收到更新的属性值。



属性复制也尊重更新频率和带宽限制，无论您可以在服务器上的每个帧更改复制的属性，只要更新了，客户端就会获得最新的值。

服务器没有义务发送每个中间值来启用属性复制。

你可以在C ++ 中添加复制说明符，你需要采取额外的步骤在actor的CPP文件中定义镀锌的生命周期复制Props函数。你也会希望在那里包括虚幻网络。



此函数用于指定应该在什么条件下复制哪个属性。在最简单的情况下，您只希望将属性始终复制到所有客户端，并通过在Do rep生命周期宏中命名该属性来实现此目的。

但您也可以指定复制的条件。例如，您可能只需要将属性复制到拥有的客户端，或者您可能只需要非欠客户端来获取最新进展。

或者您可能有一个在生成时初始化的属性，但在运行时永远不会更改。

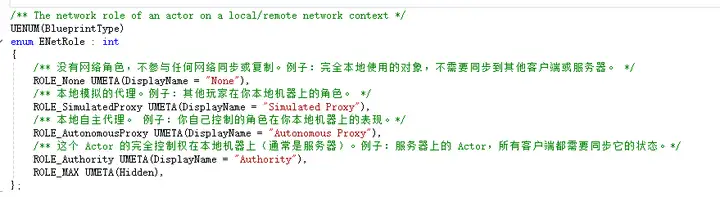
如果您需要在更新复制属性时运行一些代码，则可以声明一个rep通知函数，使用复制的使用说明符，只要由于复制更新而值发生更改，指定的通知函数将在客户端上调用。

请注意，在蓝图中，更改服务器上属性的值将自动调用服务器上相关的rep通知函数。这在C ++ 中不是这样，如果你希望你的rep通知逻辑在服务器和客户端上运行，那么你需要在更新你的属性值后手动调用rep通知函数。



## **权限和角色**

我们应该考虑最后一个概念，那就是Actor的网络角色。一个Actor可以扮演几个不同的角色，但在大多数情况下，你只需要关心一个简单得多的问题。

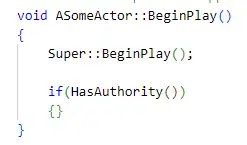


我对这个Actor有权力吗？



每当你在actor类中运行一些代码时，你可以检查授权。

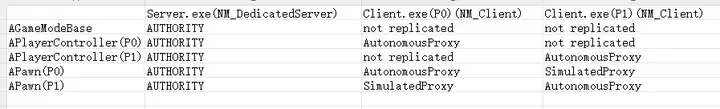
如果你有权限，你在更新actor的状态方面有最终决定权，要么是因为游戏正在单人模式下运行，要么你的代码正在由服务器执行，要么因为actor只存在于客户端上。



如果您没有权限，您的代码将在客户端上运行，并且Actor将从服务器中复制。

在这种情况下，客户端的Actor副本是服务器上权威版本的代理。

如果一个Actor没有权威，那么他的角色几乎总是被模拟的代理。通常，自主代理仅在我们谈论玩家时才进入画面，玩家控制器作为自主代理复制到拥有的客户端，而关联的pawn是该客户端的自主代理。

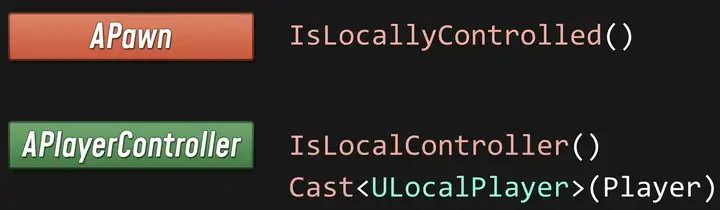


对于所有其他客户端，典当将作为模拟代理复制。自主指的是客户端直接控制Actor的动作和行为，即使它没有完全的权力。

但除非你处理的是玩家角色，否则通常只是一个二元问题。我是否有权威？



如果你正在处理与玩家相关的代码，那么另一个重要的问题是玩家是否在本地控制。



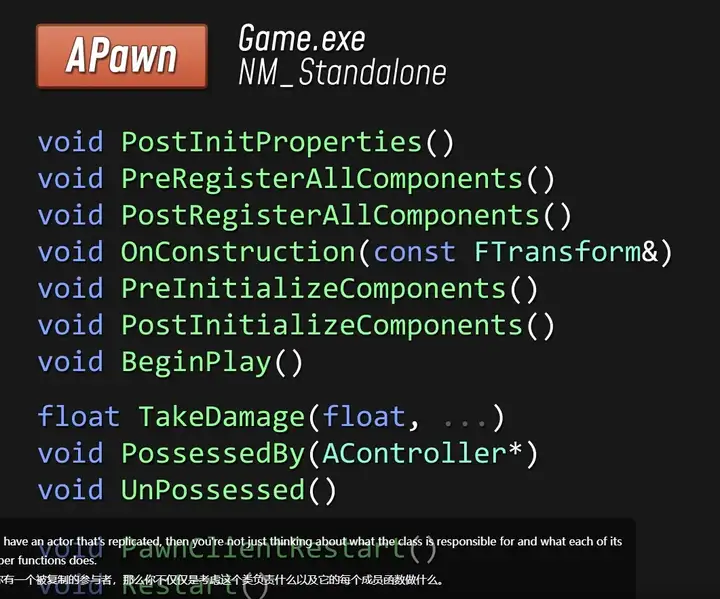
如果Pawn是本地控制的，那么该玩家对应于您的代码正在运行的游戏。如果没有，它是一个远程客户端播放器。

## **思考复制**

因此，一旦你脑子里有了所有这些概念，习文持多人游戏最重要的一点就是它增加了你思考代码的复杂性。

你可能会认为程序员把所有的时间都花在拼命地打字上，就像在电影中一样，但作为一名程序员，你很多时间都花在推理代码如何执行上，无论你有多聪明，你的大脑一次只能容纳几条信息。因此，如果你必须同时处理太多事情，编写良好、可维护的代码将非常困难。许多技能和经验归结为知道如何以相对简单的推理形式设计和实现复杂的功能。

支持多人游戏增加了你必须管理的复杂性层。



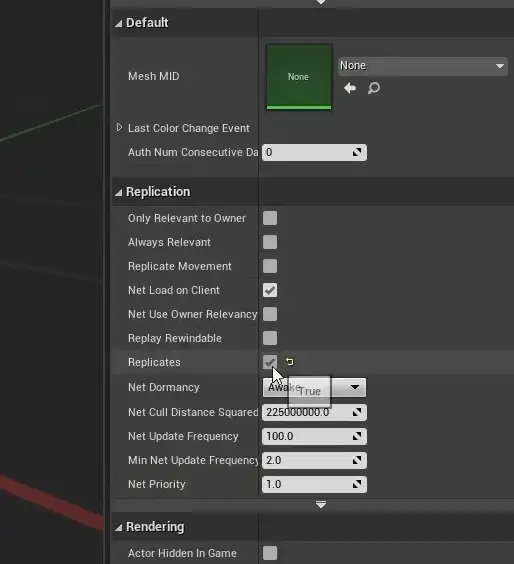
如果你有一个被复制的参与者，那么你不仅仅是考虑这个类负责什么以及它的每个成员函数做什么。你还必须考虑执行代码的位置以及数据在不同时间在网络上流动的方式，额外的复杂性一开始可能会让人感到难以承受，但随着经验的积累，它会变得更容易管理。



## **总结技巧**

总结一下，这里有一些需要记住的提示。

首先，重要的是要记住，即使一个Actor被复制，也不是它的每个部分都需要关注网络。



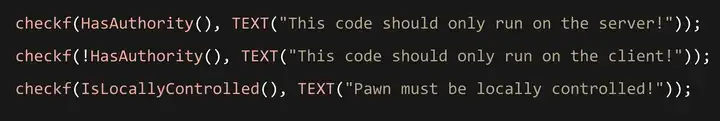
如果您有一个具有静态网格组件和动态材质实例的actor，则这些对象都不是网络感知的。你可以编写初始化和修改这些对象的函数，而不必考虑网络。

这些函数在每个游戏实例上独立运行，您只需要将复制引入到等式中，以确保它们在游戏的所有实例中以一致的方式被调用。

你的大部分游戏代码最终将基于虚幻游戏框架构建，该框架是为开箱即用的多人游戏设计的。



当您要覆盖这些类成员函数以添加自己的功能时，您需要知道一个函数是设计为仅在服务器上、仅在客户端上还是在任何地方运行。



使用断言可以从查看代码的设计运行位置明显看出，并且为帮助函数建立命名约定也很有用，例如为仅使用权限调用的函数使用像auth这样的前缀，你可以把你的参与者的功能看作是一种因果关系流程图。

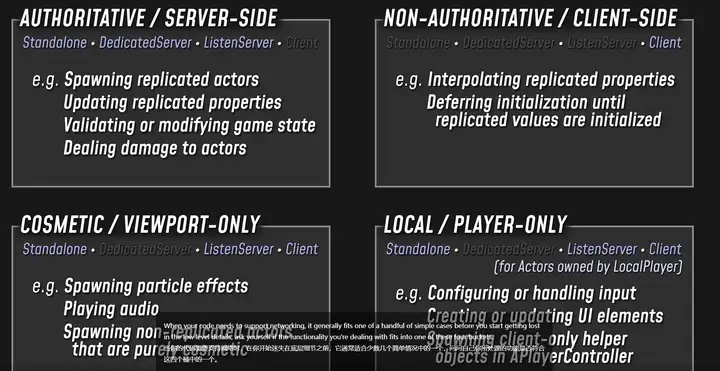


在一端，你有启动事物的事件。在另一端，您有需要发生的状态更改。因此，如果您的某个进程在网络上的不同位置启动和结束，则需要使用某种形式的复制。如果进程起源于客户端并最终在服务器上，则通常使用服务器RPC。

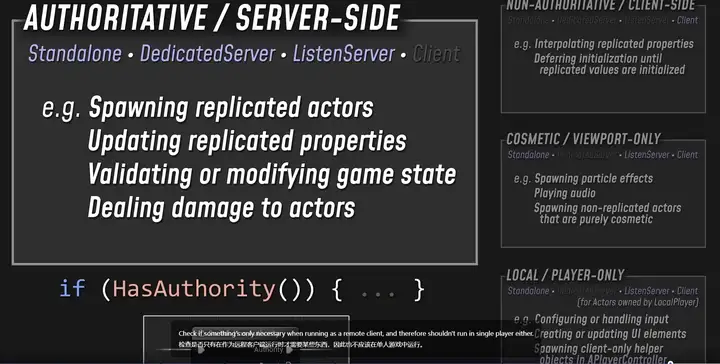


如果一个进程起源于服务器并最终对客户端产生影响，那么您需要考虑将数据复制到客户端的位置。你的代码需要一直保留在服务器上，只要它正在更新权威的游戏状态，或者如果玩家可以通过查看或修改数据客户端获得不公平的优势。但在此之后，您通常希望复制重建最终结果所需的最小数据集，并让客户端独立处理其余过程。

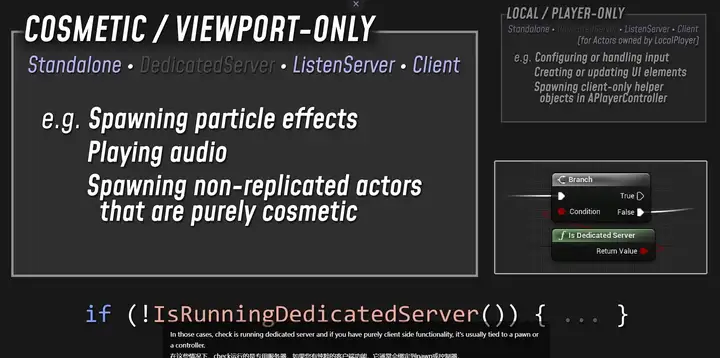
当你的代码需要支持网络时，在你开始迷失在底层细节之前，它通常适合少数几个简单情况中的一个，问问自己你所处理的功能是否符合这四个桶中的一个。



如果某些事情只需要在服务器上，在多人游戏和单人游戏中发生，请将其放在后面。它有权威。



检查是否只有在作为远程客户端运行时才需要某些东西，因此也不应该在单人游戏中运行。检查你是否没有权力。在某些情况下，您想要专门排除专用服务器。在这些情况下，check运行的是专用服务器，如果您有纯粹的客户端功能，它通常会绑定到pawn或控制器。



所以你可以检查玩家是否在本地控制。

